Week 8 Lab

For an example of how to do the exercises you can refer to my week 8 GitHub repo which is at:

https://github.com/oit-gaden/Web-Development-2019-Winter/tree/master/Examples/Week8

Don't forget to push your code to GitHub in a folder called week8.

Preparation

1. Copy your week 6 folder to make a week 8 folder. Yes, that is meant to be week6. We're going to skip week7 for this and deployment next week as there are issue with "Dockerizing" the webapp with Auth0.

Exercise 1 - Create repository for students

1. Create a "**Repositories**" folder under **webapi**.

2. Create a **repository class** and associated interface for the "**student**" entity you have defined. You can follow the code I created for
the "**product**" repository. You only have to implement the "**get all method**". You can also leave out any code related to logging.

3. You'll need add wrap the class Student (in Database/Entities/Student.cs) inside of the Database.Entities namespace. See my product.cs
for an example. You'll need to wrap the class SchoolContext (in Database/SchoolContext.cs) inside of the Database namespace. See my
ECommerceContext.cs for an example.

Exercise 2 - Create service for students

1. Create a Models folder under webapi.

2. Create a StudentDto class with the student id and email address from the student entity. Add a boolean property called "Special".

3. Create a "**Services**" folder under webapi.

4. Create a **service class** and associated interface for the StudentDto you defined in step 2. You can follow the code I created for
the "**product**" service. Again, you only have to implement the

"**get all method**". You can also leave out any code related to logging.

5. Determine some "**business rule**" for determining when to set the "**special**" property when the "**get all method**" is called. You'll be adding
   unit tests later for testing the application of this business rule. It doesn't matter how to you implement the business rule. Just that you
   add code for "calculating" the "special" property. You can follow the pattern in my BusinessRules.cs if you want.

Exercise 3 - Modify student controller

1. Replace the constructor injection of DbContext with the injection of the student service interface. See my product controller for an example.

2. Retrieve the students using the injected student service instead of the DbContext in the **GetAllStudents** action method. You will now be returning
   the **StudentDto** class rather than the Student entity class. You can also leave out any code related to logging.

3. Implement the dependency injection configuration in **Startup.cs**. Add a call to **RegisterApplicationServices** and its implementation where you
   add the dependency injection configuration for both the student repository and the student service.

4. Test the webapi with Postman as you did in week 6 to make sure it still operates as expected. Put data in the database that will trigger the setting
   of the "special" property to both true and false;

Exercise 4 - Add unit tests

1. Create a folder called **webapi.tests** under the week8 folder.

2. Change to webapi.tests and run the following command:

   dotnet new nunit

   Run the sample test by running the following command:

   dotnet test

You should see one test succeed.

3. Install test frameworks by running the following commands:

   dotnet add package FakeItEasy

   dotnet add package fluentassertions

4. Reference the project under test (webapi) by running the following command:

   dotnet add reference ..\webapi\webapi.csproj

5. Start up VSCode and add the new webapi.test project to the workspace.

6. You can remove the sample **UnitTest1.cs** file.

7. Add **StudentServiceTests.cs** (see my webapi.tests/ProductServiceTests.cs for an example)

   Create a mock student repository in the Setup method

   Create an instance of the class under test in the Setup method. In this case it is the StudentService.cs class.

   Add a test for the scenario where the "get all method" returns one or more "special" students.

   Add a test for the scenario where the "get all method" returns no "special" students.

   Use **FluentAssertion** or **NUnit Assert** for the test assertions.

8. Run the tests by running the following command:

   dotnet test

   You should see two tests pass.

9. Go into the **StudentService** class and change the criteria for determining a special student. Rerun the tests and you should see one test pass and one test fail. Go back and "correct" the**StudentService** class by returning to the original "business rule". Rerun the tests and you should now see both

tests passing.

Extra Credt (20 pts)

Add code to the webapp to modify the display of a student in some way if they are determined to be special by your business rule. Se my webapp
for an example.